Sketch2CAD: Sequential CAD Modeling by Sketching in Context

Changjian Li, Hao Pan, Adrien Bousseau, Niloy Jyoti Mitra

Sketch2CAD: Sequential CAD Modeling by Sketching in Context

CHANGJIAN LI, University College London
HAO PAN, Microsoft Research Asia
ADRIEN BOUSSEAU, Inria, Université Côte d'Azur
NILOY J. MITRA, University College London and Adobe Research

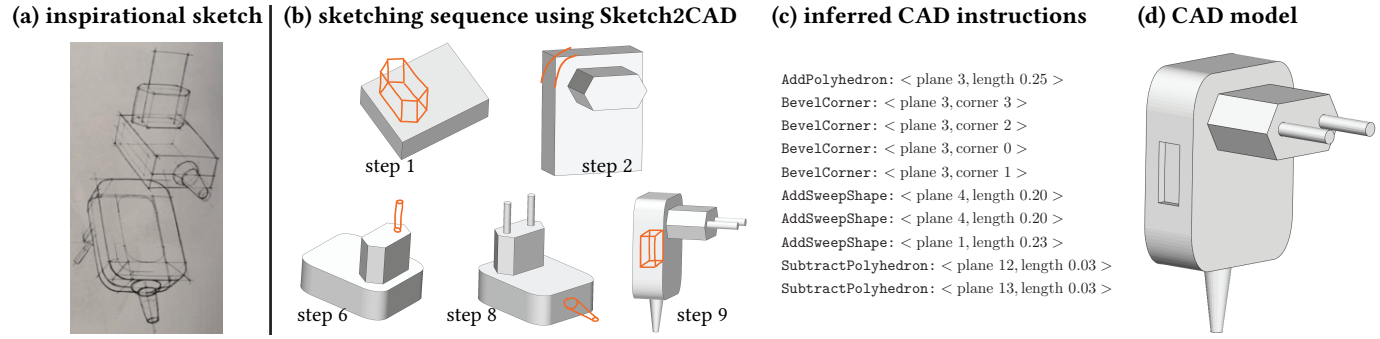

Fig. 1. Industrial designers commonly decompose complex shapes into box-like primitives, which they refine by drawing cuts and roundings, or by adding and substracting smaller parts [Eissen and Steur 2008, 2011] (a, ©Koos Eissen and Roselien Steur). Users of *Sketch2CAD* follow similar sketching steps (b), which our system interprets as parametric modeling operations (c) to automatically output a precise, compact, and editable CAD model (d).

We present a sketch-based CAD modeling system, where users create objects incrementally by sketching the desired shape edits, which our system automatically translates to CAD operations. Our approach is motivated by the close similarities between the steps industrial designers follow to draw 3D shapes, and the operations CAD modeling systems offer to create similar shapes. To overcome the strong ambiguity with parsing 2D sketches, we observe that in a sketching sequence, each step makes sense and can be interpreted in the *context* of what has been drawn before. In our system, this context corresponds to a partial CAD model, inferred in the previous steps, which we feed along with the input sketch to a deep neural network in charge of interpreting how the model should be modified by that sketch. Our deep network architecture then recognizes the intended CAD operation and segments the sketch accordingly, such that a subsequent optimization estimates the parameters of the operation that best fit the segmented sketch strokes. Since there exists no datasets of paired sketching and CAD modeling sequences, we train our system by generating synthetic sequences of CAD operations that we render as line drawings. We present a proof of concept realization of our algorithm supporting four frequently used CAD operations. Using our system, participants are able to quickly model a large and diverse set of objects, demonstrating Sketch2CAD to be an alternate way of interacting with current CAD modeling systems.

Authors' addresses: Changjian Li, University College London, 66-72 Gower Street, London, changjian.li@ucl.ac.uk; Hao Pan, Microsoft Research Asia, No.5 Danling Rd, Beijing, haopan@microsoft.com; Adrien Bousseau, Inria, Université Côte d'Azur, 2004 route des lucioles, Valbonne, adrien.bousseau@inria.fr; Niloy J. Mitra, University College London, 66-72 Gower Street, London, Adobe Research, n.mitra@cs.ucl.ac.uk.

0730-0301/2020/12-ART164 $15.00
https://doi.org/10.1145/3414685.3417807

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: sketch, CAD modeling, procedural modeling, convolutional neural network

ACM Reference Format:
Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J. Mitra. 2020. Sketch2CAD: Sequential CAD Modeling by Sketching in Context. *ACM Trans. Graph.* 39, 6, Article 164 (December 2020), 14 pages. https://doi.org/10.1145/3414685.3417807

1 INTRODUCTION

Sketching and 3D modeling are two major steps of industrial design. Sketching is typically done first, as it allows designers to express their vision quickly and approximately [Eissen and Steur 2008]. Design sketches are then converted to 3D models for downstream engineering and manufacturing, using CAD tools that offer high precision and editability [Pipes 2007]. However, design sketching and CAD modeling are often performed by different experts with different skill sets, making design iterations cumbersome, expensive, and time consuming.

While a number of methods have been proposed to create 3D models by sketching, existing solutions often lack the precision and editability of CAD modeling. On the one hand, interactive systems interpret user strokes as custom modeling operations rather than generic CAD [Bae et al. 2008; Igarashi et al. 1999; Nishida et al. 2016; Zeleznik et al. 1996]. On the other hand, methods that interpret complete sketches are limited to specific drawing techniques [Xu et al. 2014] and classes of shapes [Lun et al. 2017], and output curve networks or triangular meshes rather than editable models. As stressed by a recent survey [Bonnici et al. 2019], to be widely used by the design industry, *"sketch-based modeling systems should integrate seamlessly with existing workflow practices"*.

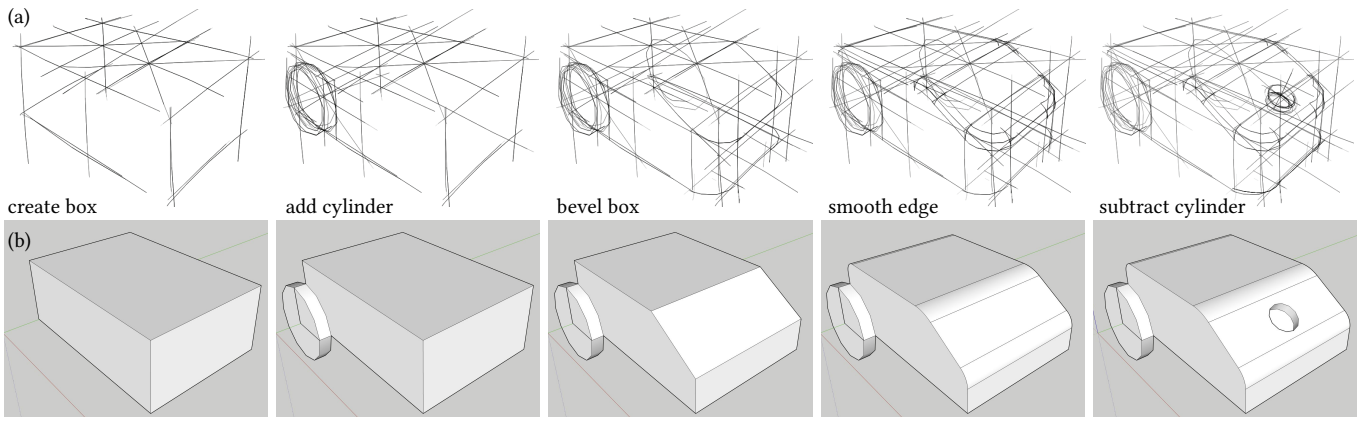

Fig. 2. **Similarity between sketching and CAD modeling workflows.** (a) As illustrated by this sequence from the *OpenSketch* dataset [Gryaditskaya et al. 2019], industrial designers construct their drawings by starting from a simple shape (here a box) that they refine by adding or subtracting sub-parts (wheels, beveled edges, hole). (b) Modern CAD software such as SketchUP [Trimble 2019] rely on very similar operations to model 3D shapes.

Our key observation is that despite their apparent differences, design sketching and CAD modeling actually involve very similar workflows, yet expressed in different languages. Industrial designers often start their sketches by drawing the overall shape as an assemblage of boxes and cylinders called *scaffolds*, which they then refine by drawing roundings, sub-parts, and small details (Fig. 2a). Similarly, CAD modelers often start with simple geometric primitives that they refine to build up complex models by progressively applying geometric operations (e.g., extrude, bevel, smooth) (Fig. 2b). Based on this observation, we propose Sketch2CAD as a learning-based interactive modeling system that translates sketching operations into their corresponding CAD modeling operations. Users of our system thus express their ideas using similar sketching steps as they would do on paper, yet obtain as output a regular CAD model, along with a trace of the sequential operations, ready to be fabricated or further edited with existing CAD software. Our system can be seen as a translator that interprets user drawn strokes in context of the current modeling session, and maps them into a sequence of predefined CAD operations, along with their associated parameters. The system empowers users to create regular CAD models without having to navigate complex CAD system interfaces.

We first propose a common parameterization of popular sketching and CAD modeling operations (e.g., `extrude`, `bevel`, `add`, `subtract`, `sweep`). For each operation, our parameterization encodes the different components of the CAD shape, which correspond to different strokes in the sketch. For instance, a `bevel` operation is composed of two parallel curves that define the new profile of the corner on which it applies. Importantly, our parameterization also encodes the faces of the current 3D model that should be modified by the operation, since CAD operations are typically applied in sequence to progressively achieve complex shapes.

The main challenge is then to recover, for every step of a sketching session, the intended modeling operation and the asociated parameters. This is a highly ambiguous task, not only because the strokes are often imprecise, but also because similar strokes might have different meaning depending on the *context* in which they are drawn. We propose a three-stage pipeline that progressively reduces this ambiguity to produce regular CAD objects. The first stage *classifies* the sketch among possible CAD operations (`extrude`, `bevel`, `add`, `subtract`, `sweep`). In addition to the user sketch, the classifier takes as input depth and normal maps of the current 3D model, which provides strong *contextual cues* about the intended operation. The second stage *segments* the user sketch and contextual maps into parts, specific to the target CAD operation. For instance, the sketch of a `bevel` operation is segmented into its two profile curves, while the contextual maps are segmented to form a mask of the face on which the `bevel` operation needs to be applied. Finally, the third stage *instantiates* the CAD operation by fitting parametric curves or shapes on the segmented strokes and projecting these strokes, optionally regularized, on the selected faces of the 3D model.

From a technical standpoint, we realize our classification and segmentation stages with deep convolutional networks. In addition to the design of CAD-specific segmentation networks, a contribution of our work resides in a large training dataset of CAD-like objects that we synthetically generated by sampling sequences of CAD operations. We took special care in balancing this dataset such that the most complex operations appear more frequently, and that parameters of all operations are sampled uniformly. Furthermore, we also balanced the length of the operation sequences, such that our system can recognize CAD operations at any stage of a modeling session.

In contrast to prior learning-based methods that were trained on particular domains [Huang et al. 2016; Nishida et al. 2016] or selected object classes [Lun et al. 2017], a key strength of our approach is that it recognizes a set of existing CAD operations that can be applied in arbitrary order, allowing the creation of a diverse range of human-made objects. In addition, the parametric nature of each such operation results in shapes that are highly precise and regular despite very approximate input strokes. Figure 1 shows a typical modeling session using Sketch2CAD. Finally, since our training is entirely synthetic, we believe that the same approach can be used to extend to support other operations.

While our sketch-based modeling system does not provide the same level of comprehensive modeling as modern CAD software

like SketchUP [Trimble 2019] and TinkerCAD [Autodesk 2019], it demonstrates an alternate way of interacting with existing CAD systems without requiring repeated command selection and switching. Our interface can be particularly attractive to product designers or novice users who are more fluent with sketching than with CAD modeling interfaces. By allowing non-experts to quickly produce complete CAD protocols (see Sec. 8.1), our tool holds the potential to facilitate more direct collaboration between novices and experts.

In summary, our main contributions are:

- formalizing a set of common CAD operations and their corresponding sketches, allowing an automatic translation between the two domains;
- developing a pipeline of deep neural networks capable of recognizing and segmenting CAD operations from sketches drawn over 3D shapes, and producing precise, regular 3D geometry by fitting CAD parameters on the predictions;
- designing a large dataset of synthetic CAD models, along with their step-by-step construction sequences; and, as a culmination of these taken together,
- presenting Sketch2CAD as a novel sketch-based modeling system that unifies the sequential workflows of product design sketching and CAD modeling.

Code, training data, and the Sketch2CAD system are available on the project page for research use.

2 RELATED WORK

Our work aims to bridge the gap between sketch-based and CAD modeling.

CAD modeling. Computer-Aided Design has long been adopted by the industry to create precise and high-quality 3D models suitable for physical simulation, lighting simulation, and downstream manufacturing [Autodesk 2019a,b; Robert McNeel & Associates 2019; Trimble 2019]. However, the high precision offered by CAD comes at the price of complex interfaces to allow users to select appropriate geometric operations and tune their parameters. Various approaches have been considered to reduce this user burden, from automatic alignment of existing CAD models on scanned point clouds [Avetisyan et al. 2019], to educational visualizations of modeling sequences [Denning et al. 2011]. We contribute to this effort by instantiating CAD operations by sequentially interpreting hand-drawn sketches.

Closer to our work are methods aiming at converting raw 3D meshes into editable CAD models, which can be formulated as a form of program synthesis [Du et al. 2018; Sharma et al. 2018; Tian et al. 2019]. On the one hand, leveraging the sequential nature of sketching and CAD modeling makes our problem better posed than the conversion of complete objects that these methods target. On the other hand, we take as input approximate sketch lines rather than precise 3D models, which induces additional ambiguity. Ellis et al. [2018] also applied program synthesis to convert sketches to graphics programs, but focused on 2D diagrams and as such did not consider depth recovery.

Sketch-based modeling. Existing work on sketch-based modeling can be broadly classified into *offline* and *online* methods. Offline methods aim at interpreting complete drawings, either automatically or with user assistance. Early algorithms detect geometric constraints between curves, such as parallelism, orthogonality and symmetry, and solve for the 3D curve network that best satisfies these constraints [Cordier et al. 2013; Lipson and Shpitalni 1996; Naya et al. 2002; Wang et al. 2009; Xu et al. 2014]. The main limitation of these methods is that they require clean drawings as input to detect and enforce relevant constraints. In addition, the curve networks they produce are not directly usable by downstream 3D modeling and simulation software. These limitations are partly addressed by interactive tools that allow users to align geometric primitives over the drawing [Gingold et al. 2009; Shtof et al. 2013]. While the parametric nature of these primitives brings robustness to approximate inputs, users of these systems need to provide a number of annotations to achieve precise alignment and relative positioning. We differ from the above methods by exploiting the common sequential nature of sketching and CAD modeling, which allows us to automatically recognize parametric CAD operations as soon as they are drawn rather than during a subsequent annotation process.

In contrast to the above optimization-based approaches, recent work has explored the potential of deep learning to automatically reconstruct 3D objects from one or several sketches [Delanoy et al. 2018; Li et al. 2018; Lun et al. 2017; Su et al. 2018]. Since these methods build strong shape priors from training data, they are limited to specific classes of objects [Delanoy et al. 2018; Lun et al. 2017; Su et al. 2018] or types of surfaces [Li et al. 2018]. Besides, all these methods predict depth maps or voxel grids that are then converted to triangular meshes, which, in contrast to CAD models, greatly limits the precision and editability of the resulting 3D models. While we also build on powerful deep convolutional networks, a strength of our approach is to predict CAD operations rather than complete objects. Combining these operations allows users to produce a wide variety of parametric shapes, which are precise and editable by construction, and allow for generalization across different CAD models.

The sequential workflow we offer makes our method closer in spirit to *online* sketch-based modeling systems, where users create complex 3D shapes incrementally by alternating between 2D sketching and 3D navigation. Because of the difficulty of recovering 3D shapes from 2D strokes, a number of systems focus on specific modeling operations, such as inflation of smooth shapes [Igarashi et al. 1999; Nealen et al. 2007] or creation of sparse networks of 3D curves [Bae et al. 2008; Schmidt et al. 2009]. Closer to our work are methods that enable the creation of CAD models representing man-made shapes. Rivers et al. [2010] resolve 2D-to-3D ambiguity by asking users to draw the shape parts in three orthographic views, as common in CAD software. We instead let users draw in a single perspective view, as common in product design sketching. The seminal *SKETCH* system by Zeleznik et al. [1996] and *GiDES++* by Jorge et al. [2003] include some of our CAD operations. However, users of *SKETCH* specify these operations using a custom vocabulary of sketching *gestures*, while users of *GiDES++* need to decompose object parts into individual strokes interpreted one by one using a set of hand-crafted rules. Our originality is to automatically recognize the CAD operations and recover their parameters from freehand

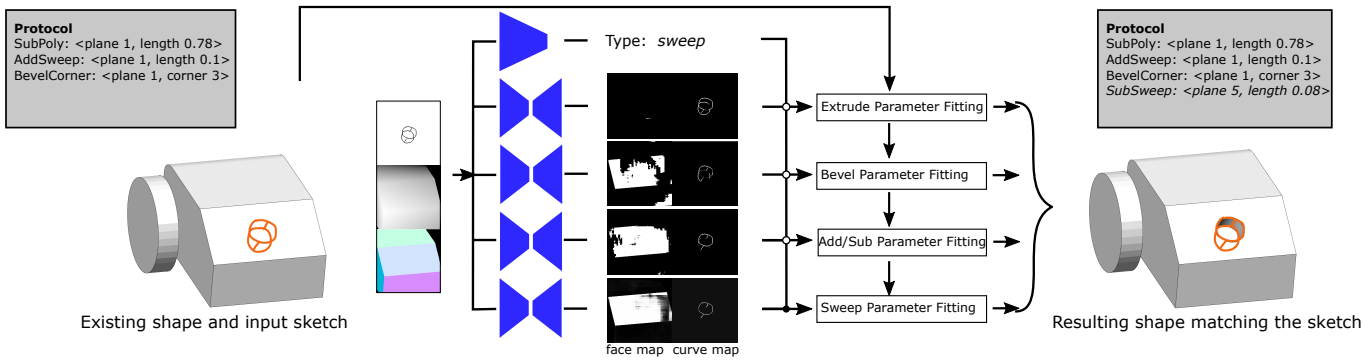

Fig. 3. **Sketch2CAD at inference time**. Given an existing shape and input sketch strokes (shown in orange) for the current operation, we first obtain the maps of sketch and local context (i.e., depth and normal), which are fed to the operator classification and segmentation networks. The classified operator type, sweep in this example, is used to select the output base face and curve segmentation maps, based on which the parameters defining the operator are fitted, via an optimization, to recover the sketched operation instance. The recovered operator is then applied to the existing shape to produce the updated model; meanwhile, the operation is pushed into the protocol list.

sketches, which allows users to directly draw complete parts of the shapes they wish to obtain, without requiring to learn a set of new gestures. We achieve this recognition using deep neural networks trained on synthetic CAD modeling sequences. While Huang et al. [2016] and Nishida et al. [2016] explored a similar usage of deep learning for procedural modeling, they target shapes with fixed numbers of parameters, created using a fixed order of operations. For example, Nishida et al. assume that to create a building, users start by sketching the building mass, then the roof, the facades, and finally the windows. In contrast, a major challenge we face is to recognize generic CAD operations with varying number of parameters, sketched in any order. We achieve this goal by accounting for the *context* under which the sketch is drawn. Furthermore, while Huang et al. and Nishida et al. use a regression network to predict the parameters of their shapes, we found this strategy to fail on the more ambiguous problem we target, and instead use a classification network to segment the sketched strokes into CAD-specific components on which we subsequently fit geometric primitives.

We draw inspiration from several earlier systems that explored the possibility to sketch novel shapes in the context of an existing scene, represented as photographs or 3D models [De Paoli and Singh 2015; Favreau et al. 2015; Lau et al. 2010; Li et al. 2017; Paczkowski et al. 2011; Xu et al. 2019; Zheng et al. 2016]. However, these method use the existing context to either guide user sketching or to deduce geometric constraints for lifting the sketch to 3D. Our originality is to leverage context within a sequential modeling workflow, where the existing scene informs the recognition of the intended CAD operation, which aims at modifying that scene.

3 METHOD OVERVIEW

Suppose we work with solid CAD models $\mathbf{M} := \{M \subset \mathbb{R}^3 | M = \overline{M}\}$, where $\overline{M}$ is the closure of M; in the implementation, we represent the models by their boundaries as triangle meshes. We define a set of CAD modeling operators $\mathbf{O} = \{O(\theta, \cdot) : \mathbf{M} \rightarrow \mathbf{M}\}$, where each applied operator $M' = O(\theta, M)$ changes the input geometry M to a new shape M' and is defined by a parameter vector θ that specifies both the parts of M to be modified and the corresponding modification parameters. Note that different operations can have different number and types of parameters. In the sketch-based CAD modeling, our primary goal is to interpret a sketch drawn over an existing shape as the corresponding operator with proper parameters that changes the shape to match the 2D sketch. The overall process is illustrated in Fig. 3. Formally, given the current shape M and the 2D sketch curves $\mathbf{S} = \{s_i\}$ with known viewpoint $\mathbf{v} \in \mathbb{R}^6$, we strive for the mapping $\Phi(M, \mathbf{S}, \mathbf{v}) = O(\theta, \cdot)$ such that the image of $O(\theta, M)$ closely matches $\mathbf{S}$ when viewed according to $\mathbf{v}$. We use the orthogonal 3D to 2D projection in our approach. In the following discussion, whenever possible, we omit the parameters of an operator for brevity.

Due to the diversity and infinite variation of operators, neither the brute-force exhaustive enumeration of all operators and parameters nor the traditional stochastic or energy based optimizations can efficiently solve the inverse problem. Instead, we approach this problem by using deep learning. In particular, we train a two-stage neural network that models the mapping Φ, where the first stage predicts the operator type and the second stage segments the sketch and context maps into regions, on which the specific parameters for the operator are fitted to instantiate the operation. The key technical challenges are how to design the machine learning models and training tasks such that the inverse mapping is feasible, learned by the neural networks and reliably generalized to real modeling interaction.

We present the definition and parameterization of specific operators in Sec. 4, the neural networks and their usage for the inverse mapping in Sec. 5, and how to train the networks for reliable generalization in Sec. 6.

4 THE OPERATORS

In the current system we support the following four operations, i.e. face extrusion, beveling a corner, addition/subtraction of a right polyhedron, and sweeping a cylindrical shape (see Fig. 4 for illustrations). We choose the four operators because they are widely used both in sketching workflows and CAD modeling, and can already be interleaved to generate complex shapes; nonetheless, our system can be easily extended to incorporate more operators as needed. To fully describe an operator $O(\theta, \cdot)$, we define its parameters θ, its

Extrude	
param:	base face f, offset d
action:	move f along its normal direction for d
sketch:	edges of the moved f and the extended side edges

Bevel	
param:	base face f, corner c with an opposite corner c', profile curve l on f
action:	turn c and c' into rounded corners specified by l
sketch:	l and its offset by vector cc'

Add/Subtract	
param:	base face f, prism base curve c, profile length d, add/subtract option $o = \pm$
action:	build a prism with base c and profile edge of length d in the normal direction of f, then find the union ($o = +$)/difference ($o = -$) of base shape and the prism
sketch:	edges of the prism

Sweep	
param:	base face f, base/offset circles c_0, c_1, profile curve c_p, add/subtract option $o = \pm$
action:	build a swept shape by rolling the profile curve c_p along c_0, c_1, then find the union ($o = +$)/difference ($o = -$) of base shape and the primitive
sketch:	circles, profiles of swept shape

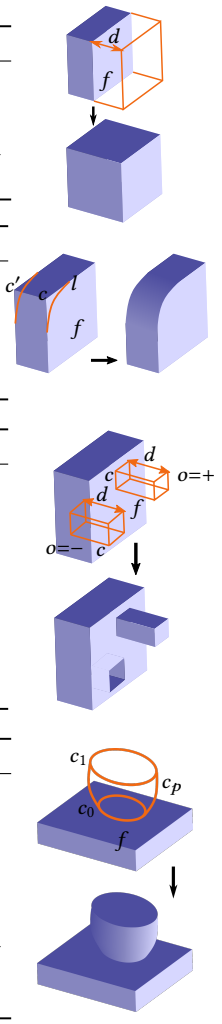

Fig. 4. **Operators supported in Sketch2CAD.** In each inset, the parameters defining the operator are annotated and the corresponding sketches are shown over the existing shape, while the result of applying the respective operation is shown as the updated shape.

applied action $M' = O(\theta, M)$ for given M, and the corresponding sketches $\mathbf{S}$ that a user draws to specify it.

`Extrude`. Extrusion is the simple offset of a planar face of the 3D shape along the face normal direction. As shown in Fig. 4, the parameters defining the operator are the face f to be offset and the distance d along face normal vector for the extrusion, with $d > 0$ for pulling out and $d < 0$ for pushing in. The corresponding sketches are the lines extruded and the boundaries of the offset face.

`Bevel`. Bevel, also known as *rounding* [Eissen and Steur 2011] in sketching or *fillet* in CAD modeling, is to turn a sharp crease of an object into a smooth and rounded connection. As shown in Fig. 4, the operator is defined on the crease connecting two corners c, c', with c residing on the base face f; the sharp edge cc' is then turned into a smooth connection with profile curve l that rounds c on f. The sketches corresponding to such an operator are the profile curve l on f and its parallel obtained by translation by cc'.

`Add/Subtract`. The addition or subtraction operator is to place a primitive shape (a prism) over a base shape and compute the union or difference of the two shapes. The parameters are used to designate the base face f to place the primitive, and to define the primitive shape by specifying its base curve c as one of triangle, quadrilateral, pentagon, or hexagon, as well as its profile curve that is always parallel to the base face normal with length d. In addition, the option o of union (for addition) or difference (for subtraction) between the base shape and the primitive is specified. The corresponding sketches are simply depicting the primitive shape by highlighting its feature curves.

`Sweep`. Sweeping a curved profile line along two circular rails is another commonly used operation in CAD modeling, which also appears frequently in industrial design sketching in the form of horizontal ellipses joined by a vertical section (see Fig. 4). Similar to `add/subtract`, the swept shape is combined with the base shape by either union or subtraction; therefore, the `sweep` operation can be seen as a special `add/subtract` where the primitive shape is a swept cylindrical shape. The parameters to define the `sweep` operation consist of the base and offset circles, and the profile curve whose two ends lie on the two circles. There is also the union and difference option to specify the combination with base shape. The corresponding sketches simply show the swept shape through its two circular ends and a pair of profile curves. The `add/subtract` and `sweep` operators are denoted as `Add/SubtractPolyhedron` and `Add/SubtractSweepShape` respectively in operation sequences (see Figs. 1, 3 and 6) for distinction.

Extension to more operators. One can follow the above examples to define new operators. In general, the parameters of the operator should be minimal but complete in defining its actions without ambiguity. The corresponding sketches should be concise and capture the important features of the operation. All these designs will impact the machine learning models used for recovering the operator instance from sketches, as discussed later in Sec. 5.

Protocols for CAD modeling. A protocol file is a serialization of the modeling steps. It consists of the full set of parameters specifying the operators that are applied in sequence to obtain the final shape. A protocol can be saved, loaded, edited, and reused for more complex modeling tasks. Illustrations of a protocol as the sequence of operations it contains are shown in Figs. 1, 3, 6. More protocol texts for generating models shown in Fig. 12 can be found in the supplemental material.

Implementation of operator actions. In our current implementation, we represent the 3D solid models by their boundaries as triangle meshes, but always maintain a set of planar polygonal faces that are made of adjacent triangles of coplanarity, by flooding across mesh edges with tight dihedral angle thresholding ($< 1^\circ$). When applying any of the operators defined above which require planar bases, the base face is selected as one of the planar polygons, and the action is carried out by computing the appropriate Boolean operation between the sketched primitive and the base mesh, using CGAL [The

CGAL Project 2020]. While for `extrude`, `add/subtract`, `sweep` the sketched primitives are clear, for `bevel`, we construct a prism whose base face is defined by connecting c and l and whose profile edge is cc' (Fig. 4), and subtract it from the base shape. In the future, when extending to operators applied on curved faces, we consider upgrading our underlying geometry representation to more flexible ones, e.g. NURBS (Sec. 8.4).

5 CONTEXT DRIVEN SKETCH INTERPRETATION

After sketching an operation in the modeling session, there are three steps taken to interpret the current sketch S: the recognition of the operator type O by an classification network, the extraction of the individual regions from the input maps by the segmentation network for the operator type O, and the recovery of parameters θ defining the specific instance $O(\theta, \cdot)$ by counting and curve fitting. Finally, the regressed operator is applied to the existing geometry to carry out the modeling intention of the user.

For all networks, the input is the concatenation of three maps, all of spatial size 256×256: the sketch map S, with $S(x, y) = 1$ for the stroke pixels (x, y) and $S(x, y) = 0$ otherwise, and the local context maps D, N encoding depth and normal, obtained by rendering the existing geometry along the sketched viewpoint $\mathbf{v}$. The viewing frustum for generating the maps is twice the size of the sketch bounding box, to ensure the user input is well covered. For the depth map, $D(x, y) \in [0, 1]$ is the normalized depth value for a foreground pixel (x, y) and $D(x, y) = 0$ to indicate background pixels. We normalize the depth map linearly such that the farthest depth is mapped to 0 and the closest depth to 1, thus removing the dynamic range of possible depth values to ease learning. For the normal map, $N(x, y) \in \mathbb{R}^3$ is the normal vector that is first transformed into the 3D camera space and then shifted by $(1, 1, 1)$ for a foreground pixel, and $N(x, y) = (0, 0, 0)$ for background pixels.

5.1 Operator classification

The classification network is a CNN with alternative layers of convolution and pooling that finally outputs the probabilities for the operator types that the input sketch represents; see supplemental material for the detailed structure. The training loss is the weighted cross entropy:

$$\mathcal{L}_{cls}(S, D, N) = -w_O \log(P_O), \tag{1}$$

where O is the ground truth operator type for the input training sample, the class weights $(w_{O'})_{O' \in \mathbf{O}}$ are computed by normalizing the inverse type frequency vector $\left(\frac{1}{N_{O'}}\right)_{O' \in \mathbf{O}}$, with $N_{O'}$ the number of training samples of type O', and P_O is the predicted probability of the operator being of type O. We use weights for different operation types to avoid potential statistical bias caused by their contrastive frequencies in the training set, as discussed in Sec. 6.

5.2 Operator regression

Rather than directly regressing the parameters, we solve the regression in two steps: first, we use deep neural networks to segment the sketch and context maps into regions corresponding to the defining structures of the operators, and second, we fit operation parameters to the detected regions using counting, searching and optimization procedures. The benefits of such a two-step regression, are that the networks are only required to learn the single-modality segmentation tasks, which is considerably more tractable than brute-force regression of diverse operator parameters, and that the parameter fitting is robust to inaccuracies of network predictions. Further, this design choice facilitates generalization across different operations. In contrast, by trying to regress directly the various parameters of an operation, we face several difficulties: to recover the extrusion and offset distances from 2D images has the inherent scale ambiguity, the number of base polygon sides of the `add/subtract` operation is changing and needs complex network structures to accommodate, and the regression of curved strokes requires fixed Bezier or spline parameterization, while in our case we can choose suitable representations to do the curve fitting.

The general network structure. Each of the segmentation networks is a U-Net that outputs two maps through two decoder branches: the probability map F of base face, and the curve segmentation map C, both of spatial size 256×256 and channel width 1; details of network structures are provided in the supplemental material. To train the network, the loss function is in the following general form:

$$\mathcal{L}_{reg}(S, D, N) = \frac{1}{256^2} \|F - \widetilde{F}\|^2 + \frac{1}{|\widetilde{M}|} \|\widetilde{M} \odot (C - \widetilde{C})\|^2,$$

where maps with $\sim$ are ground truth or precomputed maps, i.e., $\widetilde{F}$ is the ground truth base face map with $\widetilde{F}(x, y) = 1$ for foreground pixels and zero otherwise, $\widetilde{C}$ is the ground truth stroke map, and $\widetilde{M}$ is the corresponding stroke pixel mask. $\odot$ is the component-wise product, and $|\cdot|$ sums the map pixel values.

Given the predicted face map, we find the base face f by counting. To be specific, we first binarize the face map F by threshold 0.5, then render the face ID map of existing geometry $Id(x, y) \in \{f_i\}$, and finally find the face $f = f^*$ with the highest accumulated probability, computed as $f^* = \arg\max_{f_i} \sum_{Id(x,y)=f_i} F(x, y)$.

Different operators have their specific curve segmentation maps C and $\widetilde{C}$. The principle for designing the curve maps is that *the input-output pair should be a learnable mapping without strong ambiguity.* Next, we present the details of regression for each operator.

Extrude regression. We specify the ground truth extrusion curve segmentation map in this way: $\widetilde{C}(x, y) = 1$ for pixels of the offset curve, and $\widetilde{C}(x, y) = 0$ for profile curve pixels (see Fig. 4). Correspondingly, given the network predicted curve map C, we find the map of offset curve as $C_o(x, y) := (C(x, y) > 0.5) \wedge (\widetilde{M}(x, y) = 1)$, and the map of profile curves as $(C(x, y) \leq 0.5) \wedge (\widetilde{M}(x, y) = 1)$.

Having classified the pixels, we find the extrusion distance d by line search. In particular, the edges of the base face, denoted as ∂f, are extruded along normal direction $\mathbf{n}_f$ for d to match C_o. The linear search has a fine step size $\sigma = 0.0075$ and search range $[-1.5, 1.5]$, whereas the initial shapes have unit diagonal bounding box length. Note that by including the negative search range, we allow pushing the base face inside the model as well. We define the matching distance of the extruded face edges and the offset curve map, as $dist(d) := \sum_{\mathbf{p} \in \partial f} \min_{C_o(\mathbf{q})=1} \|\pi_{\mathbf{v}}(\mathbf{p} + d\mathbf{n}_f) - \mathbf{q}\|$, where $\mathbf{p}$ samples face edges uniformly by arc length, $\mathbf{q}$ ranges over image pixels, and $\pi_{\mathbf{v}} : \mathbb{R}^3 \to \mathbb{R}^2$ is the projection function of the current

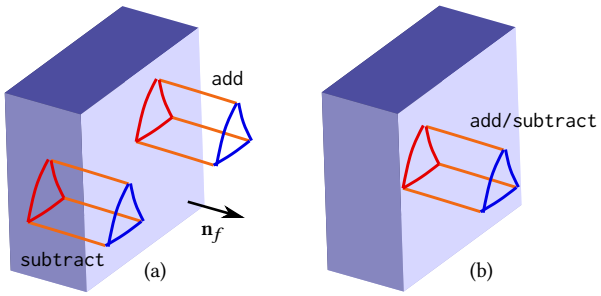

Fig. 5. **Handling ambiguity between add versus subtract.** The ambiguity of distinguishing base and offset curves for the add/subtract operator. (a) cases without ambiguity, as only one of the red and blue curves intersects with the base face. (b) ambiguous case that can be add or subtract, with the base curve being either red or blue. Instead of segmenting the base and offset curves, we regress two curves along the face normal direction (red first, blue second), thus removing ambiguity.

view. The line searched d with minimum $dist(d)$ is the regressed extrusion distance.

Bevel regression. The ground truth curve segmentation map for the bevel network encodes the two curves l and l' (see Fig. 4) in this way: $\widetilde{C}(x,y) = 1$ for pixels of l and $\widetilde{C}(x,y) = 0$ for pixels of l'. Correspondingly, we find the predicted base face curve map as $C_l(x,y) := (C(x,y) > 0.5) \wedge (\widetilde{M}(x,y) = 1)$, and the map of l' as $(C(x,y) \leq 0.5) \wedge (\widetilde{M}(x,y) = 1)$.

Assuming the profile curve is drawn in one stroke, we find the stroke corresponding to l by counting. Let s^* be the stroke with the highest accumulated probability: $s^* := \arg\max_{s_i \in \mathbf{S}} \sum_{\mathbf{p} \in s_i} C_l(\mathbf{p})$, where $\mathbf{p}$ uniformly samples s_i in the screen space. We then fit a cubic Bezier curve as l to match s^* as it is back projected onto the plane of f. Given f and l, we determine the corner c as the shared vertex of the two edges of f which intersect with l. Once we have c, the opposite corner c' is found easily.

Add/subtract regression. For the add/subtract operator, there is ambiguity with outputting the base and offset curves directly, as is illustrated in Fig. 5. To remove this ambiguity, we instead regress the two curves as ordered along the face normal direction and named the start and end curves, respectively. The ground truth curve map is given by $\widetilde{C}(x,y) = 0$ for the start curve pixels, $\widetilde{C}(x,y) = 1$ for the profile curve pixels, and $\widetilde{C}(x,y) = 2$ for the end curve pixels. Therefore, we obtain the predicted maps for the start curve $C_s(x,y) := (C(x,y) \leq 0.5) \wedge (\widetilde{M}(x,y) = 1)$, the profile curve $C_p(x,y) := (0.5 < C(x,y) \leq 1.5) \wedge (\widetilde{M}(x,y) = 1)$, and the end curve $C_e(x,y) := (C(x,y) > 1.5) \wedge (\widetilde{M}(x,y) = 1)$.

The add/subtract operation has more complex parameters than extrude or bevel (see Fig. 4), the recovery of which also involves more steps: we first classify the strokes according to the predicted curve map, then fit 2D curves to the strokes and determine the add/subtract option, and finally back project the base curve to the 3D base face and recover the prism length by line search.

Again, we classify strokes by pixel counting. For a stroke s_i, let its likelihood of being starting curve as $L_s(s_i) := \sum_{\mathbf{p} \in s_i} C_s(\mathbf{p})$, where $\mathbf{p}$ samples s_i uniformly, and similarly we have $L_p(s_i)$, $L_e(s_i)$ for the likelihood of being profile and end curves, respectively; the curve type of s_i is the one with largest likelihood.

We assume each of the profile curves is drawn by one stroke; therefore the number of profile strokes tells the N-gon of the prism base. We then find the end points of the profile curves grouped into the beginning set and the end set, which are used as the initial guess for fitting N-gons to the starting and ending strokes through iterative closet point method, respectively.

To determine the add/subtract option o, we check the intersections of fitted polgyons with the base face f. If the start polygon P_s intersects f, we have $o = +$ the addition. Otherwise if the end polygon P_e intersects f, we have $o = -$ the subtraction. However, if none of the two intersects with f, the sketch is regarded erroneous with no matching operator instance. The user is alerted with this failure. For the ambiguous case shown in Fig. 5(b), the above procedure implies the default addition option, and the user can switch it manually if needed (Sec. 7).

Finally, the 2D base polygon, i.e., P_s for addition and P_e for subtraction, is back projected onto the plane of f to obtain the 3D prism base polygon, and the prism length d is obtained by line searching the 3D base polygon along normal direction to match the pixels of the offset end. The line search process replicates that for extrusion.

Sweep regression. Since the sweep operator is a special case of the add/subtract operator, the curve maps are the same for both operators. The fitting of parameters is much like add/subtract operator as well, with minor differences in curve fitting. Note that we restrict the sweep operations to circular cross sections.

The start and end curves are fitted as ellipses to the corresponding curve maps. After having determined the base curve and add/subtract option, the ellipse is back projected to the base face plane as the base circle c_0 (see Fig. 4). The offset d, defined as the distance between the two circle centers, is again found by line search as done in extrusion, except to match the base circle center to the offset curve center in this case. The offset circle c_1 is then obtained by back projecting the offset ellipse to the translated base face plane by distance d.

To recover the profile curve, we first determine the 3D plane it lies on. For one of the 2D profile strokes, it has an intersection point with each of the two ellipses. The intersection points are lifted to 3D following the ellipse-circle back projection. The centers of c_0, c_1 and any of the two intersection points together determine the 3D plane that the profile curve resides in. We fit a cubic Bezier curve inside the plane to the profile stroke points, to obtain the profile curve. Finally, if we detect that the profile curve to be nearly linear and that the two circles have similar radii, we rectify the swept shape to be a cylinder.

6 TRAINING DATA GENERATION

To train the networks for robust performance on real sketching interactions, we need a large-scale data set that covers the possible variations. Thanks to the procedural nature of CAD modeling, we can generate the training data by synthesizing diverse procedures (Fig. 6). The training data generation therefore consists of two steps, the modeling sequence generation and the sketch image rendering, as discussed below.

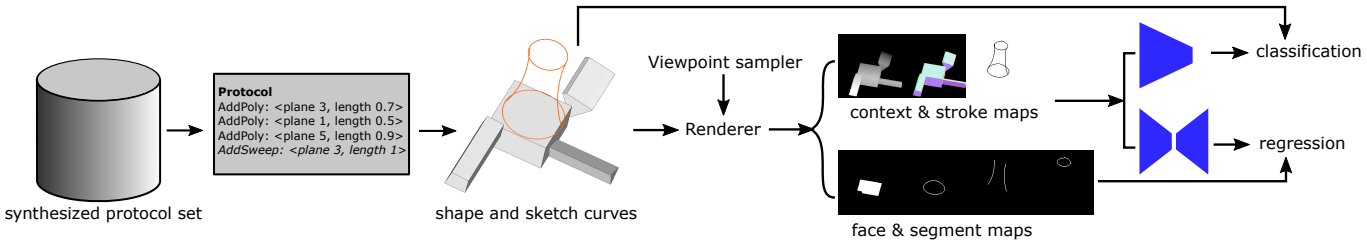

Fig. 6. **Sketch2CAD at training time.** We synthetically generated 10k protocols of diverse lengths for procedurally generating 40k training shapes. For each protocol, we execute it up to the last operation, for which the sketch curves are built and overlaid on the built shape. The sketch curves and existing shape are rendered in proper viewpoints to generate the input sketch and local context maps, as well as the ground truth face and curve segmentation maps, which are used to train the operator classifier and the corresponding segmentation network.

Sequence generation. Given a set of CAD modeling operations $\{O_i\}$, we generate training data that allows the network to learn to infer from 2D sketches the corresponding operations robustly, while avoiding the prohibitive enumeration of the infinite space of all possible 3D models and configurations of operations. Our key observation for achieving this goal is that while in theory one part of a 3D model can potentially be connected with every other part of the model, it is the local context that influences the part geometry the most and therefore provides the dominant cue for interpreting its 2D sketch properly. Based on this observation, we only need to extensively enumerate the local combinations of different operations producing diverse model variations to train the network. Thus in practice, for each operation O, in addition to its own parametric variations, we search for a sequence $\{O_1, \cdots, O_m\}$ of random operations, that are applied before the operation, i.e., $O \circ O_m \circ \cdots \circ O_1$, to simulate the local context variations. Indeed, we find that with $0 \leq m \leq 3$, there can be very complex combinations and shapes generated; some examples are shown in Fig. 7.

To balance complexity, for each sequence length $m+1 \in [1, 2, 3, 4]$, we generate 10k protocols, thus 40k shapes in total. For sequences of each length, the last operator O has a fixed frequency for different types, i.e., $1 : 1 : 4 : 2$ for `extrude`, `bevel`, `add/subtract` and `sweep`; the ratios are chosen to account for the different complexities of the four operators, allocating more samples for `add/subtract` and `sweep` which have more degrees of freedom. In addition, we generate 10k protocols with the same distribution for testing the networks. Note that since we weight the different operation types by their inverse frequencies in the dataset (Eq. 1) for training the classification network, such a non-uniform distribution does not cause bias for operation recognition.

While always starting with a base box shape, we randomize each operation instance in a synthesized sequence to cover sufficient geometric variations while avoiding degeneracy. This includes, for example, selecting a random planar face from the existing shape as the base face, applying offsets sampled in a large range, generating base polygons or circles that have centers positioned randomly inside the base face region, and polygons and profile curves that are perturbed without self-intersection.

Sketch rendering. We design the rendering process to mimic how real sketch drawing looks like. To render the corresponding sketch and context maps of an operator in the generated sequence, we randomly sample informative views around the base face of the operator, with view directions forming angles in the range of $[20°, 80°]$ with the face normal. The viewing frustum is centered around the sketch curves, and further scaled by a random factor in range $[1.6, 2.4]$ to create different zooming effects. We filter out the views where for the extrusion operation, the offset curve is occluded for more than 20%, or for the other operations, the base curve is occluded for more than 20%, as such viewpoints are unnatural to take in real sketching. The 3D curves of an applied operator instance are first projected onto the 2D camera space, then perturbed at the endpoints randomly by a Gaussian noise, and finally smoothed a little for regularization, which reproduces the style of rough freeform sketching.

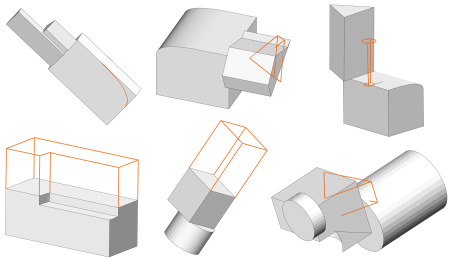

Fig. 7. **Procedurally generated training set.** Sample synthesized shapes and next step sketch by randomized combination of operations. Within only four steps, very complex shapes can already be created.

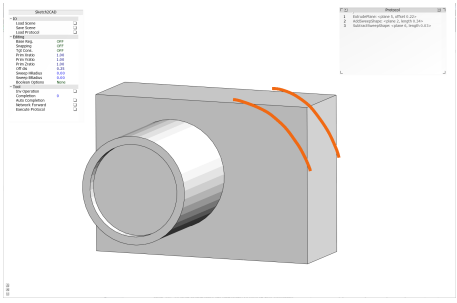

Fig. 8. **Sketch2CAD UI.** A screenshot of our prototype implementation. The tool features freeform and interactive sketching over a 3D shape in the central canvas, the editing of recovered operation parameters on the left, and the illustration of the operation sequence on the right.

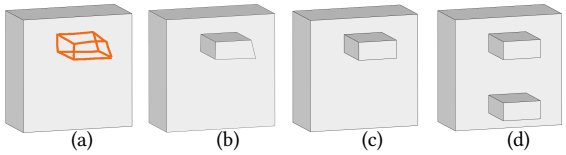

Fig. 9. **Stroke regularization and auto-completion in Sketch2CAD.** (a)-(c): the corners of the sketched quadrangle are detected to be close to right angles, and automatically regularized to form a rectangular base polygon. (c)-(d): auto-completion by reflecting the primitive against the horizontal cross section of the base shape.

7 THE MODELING SYSTEM

We build a prototype modeling tool to demonstrate our approach. The tool features interactive modeling with an user interface that allows sketching in 2D and instant feedback viewed freely in 3D. A screenshot of the our tool is shown in Fig. 8. Please refer to the supplemental video for real time sketching and modeling sessions.

Besides sketching, the tool allows the user to save, load, replay and edit the sequence of operations stored in protocol files, thus fully demonstrating the power of the procedural CAD modeling paradigm. To assist the easy sketching of *precise* CAD models, our system also implements techniques like the regularization of sketched curves, the tuning of operation parameters, and auto-completion by replicating sketched primitives through symmetry, as detailed next.

Regularization. In addition to the inherent regularization enabled by casting sketch into predefined operations in the procedural language level, our interface applies curve-level regularization, like snapping and rectification, to assist user sketching, as is commonly found in CAD modeling software. The general idea of snapping is to detect key points, e.g., centers, edge middle points, corners of the base face, and align the corners and centers of the sketched primitive shape with them, whenever the point pair comes within a (default) small distance. The general idea of rectification is to detect the approximate parallelism between sketched edges and base face edges, as well as the approximate equality of corner angles/edge lengths of sketched N-gons, and enforce the parallelism and equality by constructing parallel edges and regular N-gons analytically.

In particular, the snapping happens when the distance between the nearest key point pairs is within 10% of the diameter of the base face. The rectification happens when the differences of edge angles from zero, or of corner angles from $\frac{(N-2)180^\circ}{N}$, are within 20°, for parallelism and corner equalization, respectively. It also happens when the differences of side lengths are within 20% of the average length for side length equalization. An example of regularization for rectangular prism addition is shown in Fig. 9. The user can switch off the auto-regularization to sketch arbitrary shapes.

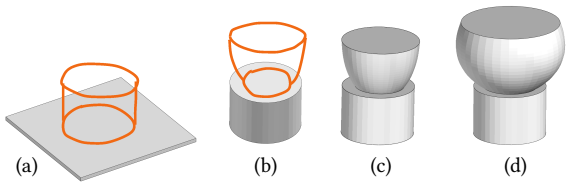

Fig. 10. **Tuning the parameters of operations in Sketch2CAD**. (a)-(b): a cylinder is sketched onto the base box, but only the cylinder is kept. (b)-(c): a swept shape is added. (c)-(d): the offset distance between the two circles of the swept shape, as well as the top circle radius, are enlarged by tuning their parameter values.

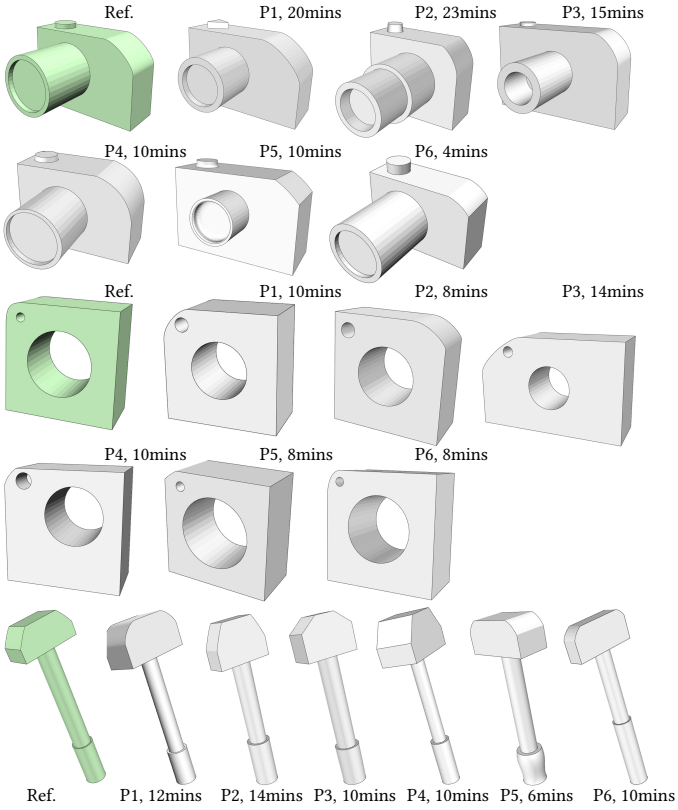

Fig. 11. **User gallery.** We asked 6 participants to reproduce 3 reference shapes, shown in green. All participants completed these modeling tasks in 5 to 20 minutes and achieved a close match to the reference.

Tuning operation parameters. As an advantage of inferring CAD operations the users can edit the recovered parameters of a sketched operation. We support three types of adjustments: creation of base shapes, resolution of ambiguous results, and fine tuning for geometric precision. First, users can select Boolean operations between existing shape and the sketched primitive, which is useful for quickly creating a base shape different from the plain box that the system starts with. Second, users can switch between the union and difference options for `add/subtract` and `sweep`, which have inherent ambiguous cases that require user specification (Sec. 5). Third, users can fine-tune the geometric parameters, e.g., offset distances, circle radius, etc. In particular for a swept shape, when tuning the distance between circles or the radius of a circle, we adjust the control points of the profile cubic Bezier curve in proportion, defined by the distance from a control point to the fixed base circle or rotational axis, to preserve the overall shape of the swept geometry as much as possible. An example of editing parameters of sequential operations is shown in Fig. 10.

Auto-completion by symmetry. Symmetry is prevalent in CAD models and can greatly ease user interaction. In our tool, the user can take advantage of symmetry by reflecting a sketched primitive

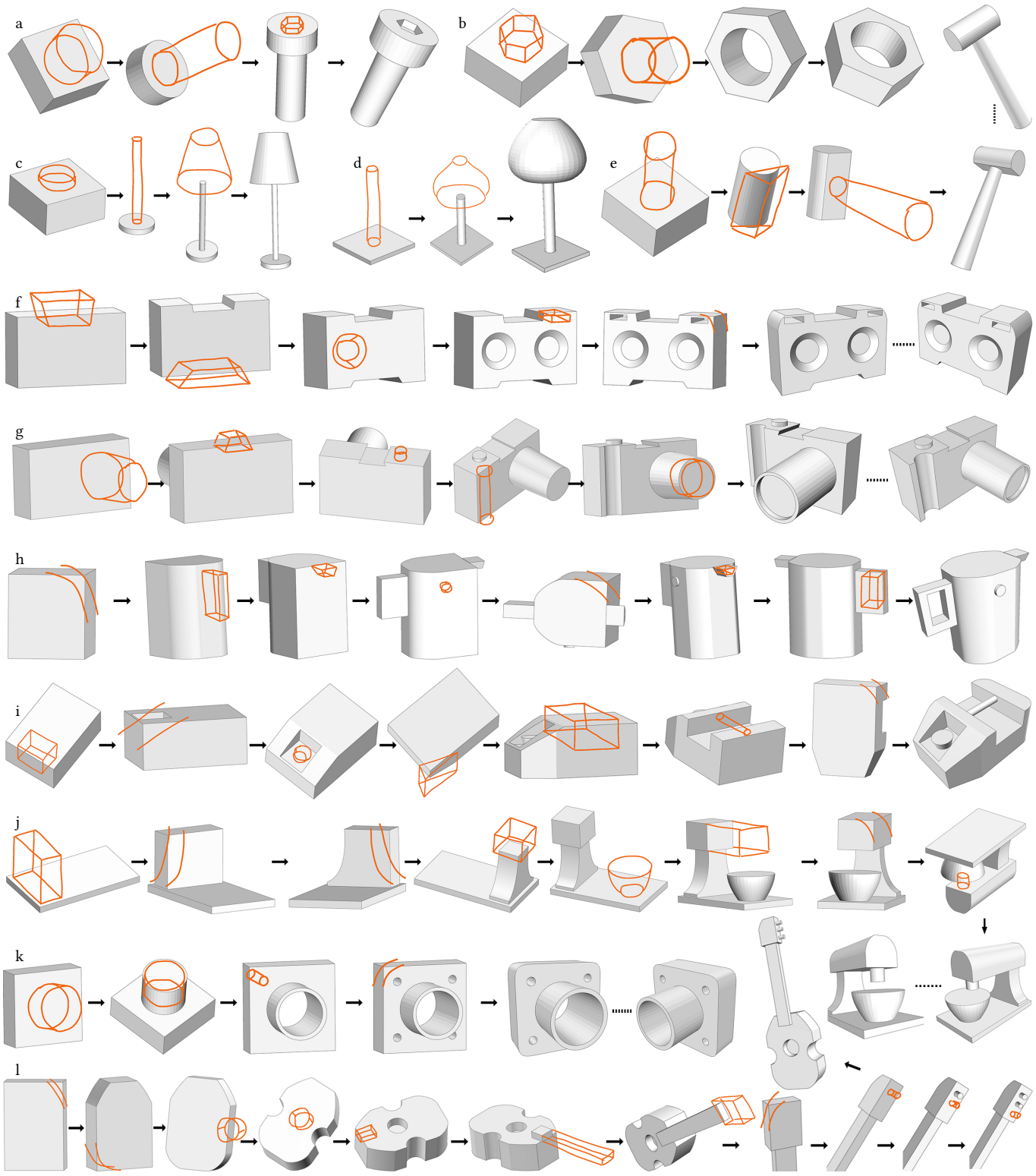

Fig. 12. **Result gallery.** Various modeling sequences created during design sessions using Sketch2CAD. The corresponding protocol steps are shown in the supplemental material. Please also refer to the supplementary video.

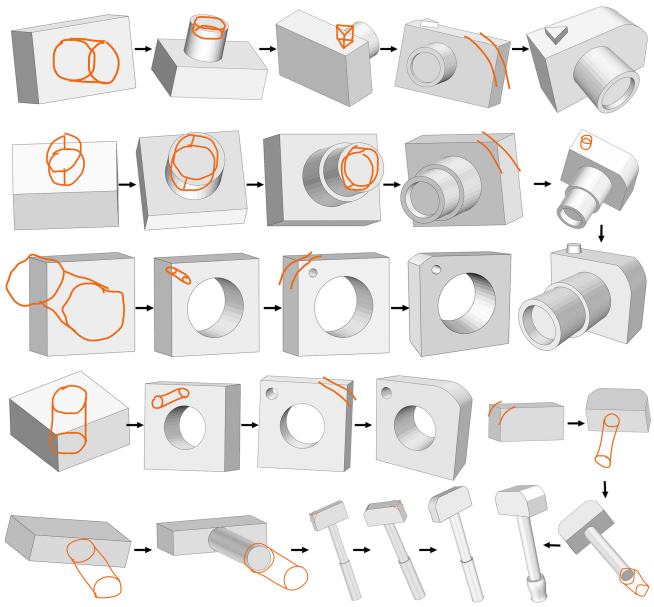

Fig. 13. **Selected user modeling steps.** Users envision different paths and variations of operations for reaching the similar targets. The freehand inaccurate sketches are robustly translated into intended operations.

shape [Peng et al. 2018] and its Boolean operation through the selected cross section planes of the oriented bounding box of the base shape, thus avoiding the need to repeat the sketch multiple times manually. An example of auto-completion by symmetry is shown in Fig. 9. More examples are shown in Figs. 1 and 12.

8 RESULTS AND DISCUSSION

With our tool, we have sketched several models of different complexities. Examples are shown in Figs. 1 and 12, with the operation sequences ranging from 2 to 11 steps, constructing CAD models from the simple bolts and nuts to the sophisticated mixer and cameras. User evaluation also confirms the ease of sketching CAD models with our approach (Sec. 8.1). We also validate the important design choices in our framework through ablation tests in Sec. 8.2, and discuss limitations and future work in Sec. 8.4. Interactive modeling sessions, complete user evaluation data, model mesh files and sample protocol files can be found in the supplemental material.

Runtime. Tested on a desktop PC with Intel(R) Core i9-9900 3.1GHz CPU and NVidia RTX 2070 Super GPU, the network inference is instantaneous, taking around 0.07s. Most time is spent on line search, ranging from 0.01s to 2s, as the step involves repeated computation of distances between pixels and stroke points, although it can be largely parallelized. To apply the recovered operator, a Boolean mesh operation typically takes around 0.02s.

8.1 User evaluation

We have evaluated the ease of use of our system by asking 6 novices [1] to create the same 3 reference shapes. The target shapes, pre-modeled by an expert user, were presented to participants as a static image (Figure 11, green). Nevertheless, we do allow the participants to try and explore with variations, so that novel and interesting deviations from the references can be expected. All participants had little to no experience in sketching nor in CAD modeling, and were given a tutorial and a short practice session to get familiar with our system.

Figure 11 shows all models created by the participants, along with their time to completion. On the one hand, all participants managed to quickly produce models that closely match the reference shapes, demonstrating the ability of our system to make CAD modeling accessible to non-professionals. On the other hand, several participants also decided to deviate from the reference, for instance by adding a second part to the lens of the camera (P2), or by modeling a curved handle for the hammer (P5). We see this unexpected behavior as a consequence of the joy and artistic freedom offered by freehand sketching.

Figure 13 provides a selection of intermediate sketching steps performed by the participants, which shows that our system is capable of interpreting a wide variety of strokes representing similar shapes. Note that all participants used a mouse to draw the input strokes, which our system nevertheless translates into regularized CAD operations. Several participants commented that they appreciated the ability of our system to produce regular shapes from approximate strokes, and that they prefer to let the modeling flow going rather than revise what they had drawn. Although given the option, none of the user turned off the stroke regularizer option in Sketch2CAD. Participants gave an average rating of 4.75 on a 5-point Likert scale when asked whether the sketches are properly translated to CAD operations, and an average rating of 5 for ease of conception of the modeling sequences. Complete user feedback and comments on the ease of use of both the sequential modeling paradigm and our prototype tool can be found in the supplemental material.

8.2 Ablation study

We validate two key components of our framework by ablation tests evaluated on the segmentation tasks for all operations:

(1) Using *local context* versus using sketch only. The comparison is shown in Table 1, where the 'no context' configuration uses only the sketch map as network input. It is clear that

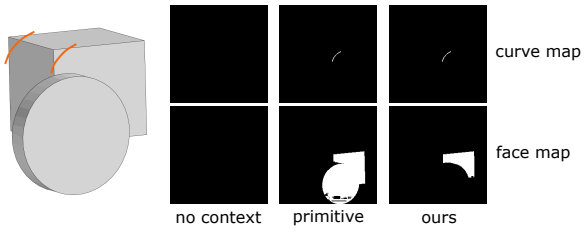

Fig. 14. **Comparing the ablation configurations by example.** The real bevel sketch is shown on the left. The 'no context' network fails to produce any output that is above the map threshold (Sec. 5). The 'primitive' network predicts a good curve map, but cannot distinguish the two front facing polygons for base face map, which leads to the wrong base face detected by counting (Sec. 5). Our full network gives almost perfect base face and curve segmentation maps.

[1] Due to requirement for GPU at inference time and restriction on lab access, we could not test the system with a wider set of users.

Table 1. Ablation tests on using context as input and using operator composition to generate training data. Our full network using context as input and trained on synthesized shapes composed by multiple operators has the best accuracy for all segmentation tasks.

Operator	Config	Face IoU(%)	Curve Acc. (%)
Extrude	no context	59.31	98.20
	primitive	70.27	87.78
	ours	**91.78**	**98.04**
Bevel	no context	3.23	50.24
	primitive	73.27	93.88
	ours	**91.73**	**98.12**
Add/Sub	no context	10.27	66.21
	primitive	63.81	87.32
	ours	**78.26**	**93.54**
Sweep	no context	7.34	48.90
	primitive	68.72	90.10
	ours	**78.34**	**95.57**

without the local context maps of depth and normal of existing shapes, the segmentation of both the base face and the sketch curves becomes very difficult, with the base face IoU frequently under 10%. In real user sketching, the networks without context are barely usable (Fig. 14).

(2) Using shapes composed by multiple operations for network training, versus using primitive shapes only. The configuration of 'primitive' shown in Table 1 trains the regression networks on another set of 40k synthesized shapes and sketches, which however only contains sequences of length 1 (Sec. 6). The 'primitive' networks are then evaluated on the same 10k testing dataset of different sequence lengths (Sec. 6) and compared with our results. It is clear that the primitive networks that do not see sufficiently complex combinations of operation cannot match the accuracy of our results, with base face IoU lower for more than 10%. Real tests by user sketching show the difference as well (Fig. 14).

In addition, we note that for the operator classification task, since the four operations have quite different sketch patterns, the two ablated configurations can achieve comparable performances as our full network does, i.e., 99.80% of no context, 93.68% of primitive and 99.79% of ours, since depth and normal maps do not play the essential role in operation classification.

8.3 Robustness of network predictions

We test the robustness of network predictions under increasing levels of sketch irregularity. While it is difficult to collect large amounts of real user sketches with different levels of irregularity, we simulate the variations by adding perturbations to clean sketches, as done for the synthetic training data generation (Sec. 6). To be specific, we add stronger stroke perturbations than the training data generation configuration, i.e., 1.4% of the rendered image diagonal length for level 1(ours), 2.8% for level 2 and 4.1% for level 3 (see Fig. 15), and evaluate how the pretrained model works under such out-of-distribution settings. The statistics are reported in Table 2 and example sketches are shown in Fig. 15. Quantitatively, as the noise increases, the segmentation networks produce more inaccurate results, and the same observation is found from the classification network (99.79% of level 1 (ours), 85.06% of level 2, and 53.20% of level 3, respectively). Qualitatively, while the segmentation network was trained with a low level of noise, it produces high-quality segmentation maps for moderate noise (level 2). While high noise (level 3) degrades the segmentation maps, the subsequent parameter fitting still yields a reasonable shape.

Table 2. Quantitative robustness test of network predictions. Both the face IoU and curve regression accuracy drop noticeably as the noise increase.

Operator	Config	Face IoU(%)	Curve Acc. (%)
Extrude	level 1(ours)	**91.78**	**98.04**
	level 2	89.59	96.28
	level 3	82.58	91.69
Bevel	level 1(ours)	**91.73**	**98.12**
	level 2	85.90	94.89
	level 3	75.76	90.27
Add/Sub	level 1(ours)	**78.26**	**93.54**
	level 2	77.09	92.02
	level 3	73.88	86.08
Sweep	level 1(ours)	**78.34**	**95.57**
	level 2	77.31	94.44
	level 3	75.47	92.78

8.4 Limitations and Future Work

In its current form, Sketch2CAD does not support drawing primitives on curved faces (e.g., on the curved face of a cylinder). One possibility would be to use NURBS as the modeling primitives, where stitching face can be curved NURBS patches stopping at trim lines. This would, however, require an extension of the underlying geometry engine used in our implementation. Another limitation involves drawing small features (e.g., knobs, or screw threads). While we do support zoom in our interface, having a library of small leaf-level part features can be useful to instantiate, rather than build up from scratch. Finally, we expect a certain amount of sketching ability from the user. Porting our code to a tablet interface can further lower this entry bar.

Our training data generation only considers geometric feasibility rather than semantics, e.g., not all combinations of the operations are

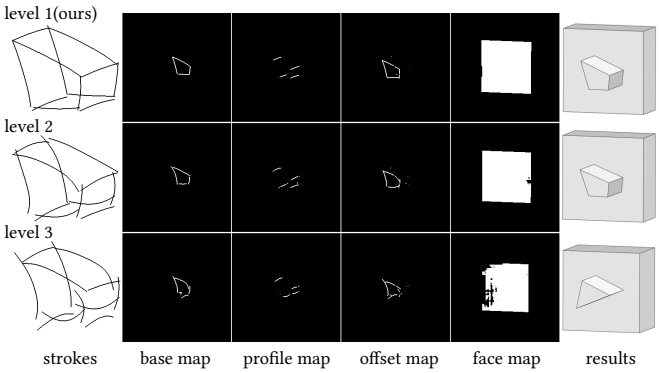

Fig. 15. **Sketch perturbation examples and the corresponding network predictions and fitting results.** Each row shows an example of a specific stroke perturbation level, while different columns show strokes, network outputs and the final result after the parameter fitting.

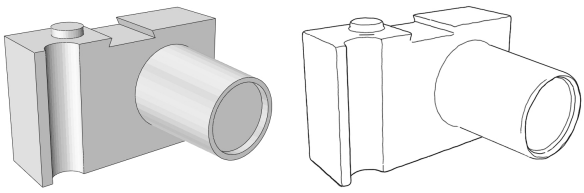

Fig. 16. **CAD-based sketching.** The CAD models generated in our system can subsequently be procedurally edited – subdivided and smoothed in this example – and the resultant mesh be used to go back to a 'sketch' using NPR rendering. This allows the user to perform operations that are much easier in the CAD domain and then transition back to sketching, possibly with camera view changes. This can be useful during ideation and prototyping phases of product design.

functionally meaningful. On the other hand, in the scenario of CAD modeling, there are strong semantics about the desired forms and functions of the different parts and their composing operations for common man-made objects. In the future, we plan to take this factor into consideration and train our networks on more realistic data that respect real world model distributions, e.g., by utilizing dataset with semantic annotations like PartNet [Mo et al. 2019]. It will also be interesting to train our network directly on CAD modeling trace data, when available, to capture typical sequences of operations and learn auto-complete routines (cf., [Peng et al. 2018]) directly from user data. Finally, while in this work we explored sketch-to-CAD, we can easily use the generated models, possibly after 3D based editing and manipulations, to go back to the sketch domain and thus enable powerful edits to sketching. Figure 16 shows an early example of such a possible workflow.

9 CONCLUSION

The visceral and approximate nature of freehand sketching is often considered to be in contradiction with the tediousness and rigidity of 3D modeling. Yet, we observed that industrial design sketching and CAD modeling follow very similar workflows, where practitioners create complex shapes as a sequence of simple sketching (resp. modeling) operations. By identifying and parameterizing common operations in the two domains, and training a deep neural network to recognize and segment these operations, we offer an interactive system capable of turning approximate sketches of human-made objects into regular CAD models, as illustrated by our evaluation with novices as well as the diversity of shapes we created with our approach.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their valuable and detailed suggestions, the user evaluation participants and Nathan Carr, Yuxiao Guo, Zhiming Cui for the valuable discussions. The work of Niloy was supported by ERC Grant (SmartGeometry 335373), Google Faculty Award and gifts from Adobe, and the work of Adrien was supported by ERC Starting Grant D3 (ERC-2016-STG 714221), research and software donations from Adobe. Finally, Changjian Li wants to thank, in particular, the endless and invaluable love and supports from Huahua Guo over the tough time due to COVID-19.

REFERENCES

Autodesk. 2019a. *3ds Max.* https://www.autodesk.com/products/3ds-max/overview
Autodesk. 2019b. *Maya.* https://www.autodesk.com/products/maya/overview
Autodesk. 2019. *TinkerCAD.* https://www.tinkercad.com/
Armen Avetisyan, Manuel Dahnert, Angela Dai, Manolis Savva, Angel X. Chang, and Matthias Nießner. 2019. Scan2CAD: Learning CAD Model Alignment in RGB-D Scans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proc. ACM UIST.* ACM, 151–160.
Alexandra Bonnici, Alican Akman, Gabriel Calleja, Kenneth P Camilleri, Patrick Fehling, Alfredo Ferreira, Florian Hermuth, Johann Habakuk Israel, Tom Landwehr, Juncheng Liu, et al. 2019. Sketch-based interaction and modeling: where do we stand? *AI EDAM* (2019), 1–19.
Frederic Cordier, Hyewon Seo, Mahmoud Melkemi, and Nickolas S. Sapidis. 2013. Inferring Mirror Symmetric 3D Shapes from Sketches. *Computer Aided Design* 45, 2 (Feb. 2013), 301–311.
Chris De Paoli and Karan Singh. 2015. SecondSkin: Sketch-Based Construction of Layered 3D Models. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 34, 4, Article 126 (July 2015), 10 pages.
Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A Efros, and Adrien Bousseau. 2018. 3D Sketching using Multi-View Deep Volumetric Prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018), 21.
Jonathan D. Denning, William B. Kerr, and Fabio Pellacini. 2011. MeshFlow: Interactive Visualization of Mesh Construction Sequences. *ACM Trans. Graph.* 30, 4, Article Article 66 (July 2011), 8 pages.
Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. 2018. InverseCSG: Automatic Conversion of 3D Models to CSG Trees. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia* 37, 6 (2018).
Koos Eissen and Roselien Steur. 2008. *Sketching: Drawing Techniques for Product Designers.* Bis Publishers.
K. Eissen and R. Steur. 2011. *Sketching: The Basics.* BIS.
Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. 2018. Learning to infer graphics programs from hand-drawn images. In *Advances in neural information processing systems.* 6059–6068.
Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2015. Line Drawing Interpretation in a Multi-View Context. In *Proceedings of the Conference on Computer Vision and Pattern Recognition.* IEEE.
Yotam Gingold, Takeo Igarashi, and Denis Zorin. 2009. Structured Annotations for 2D-to-3D Modeling. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 28, 5 (2009).
Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Frédo Durand, and Adrien Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Trans. Graph. (SIGGRAPH Asia)* 38, 6 (November 2019).
Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. 2016. Shape Synthesis from Sketches via Procedural Models and Convolutional Networks. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 22, 10 (2016), 1.
Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. *SIGGRAPH* (1999).
Joaquim A Jorge, Nelson F Silva, and Tiago D Cardoso. 2003. GIDeS++: A Rapid Prototyping Tool for Mould Design. In *Proceedings of the Rapid Product Development Event RDP.*
Manfred Lau, Greg Saul, Jun Mitani, and Takeo Igarashi. 2010. Modeling-in-context: user design of complementary objects with a single photo. In *Proc. Sketch-Based Interfaces and Modeling.*
Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2018. Robust flow-guided neural prediction for sketch-based freeform surface modeling. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)* (2018), 238.
Yuwei Li, Xi Luo, Youyi Zheng, Pengfei Xu, and Hongbo Fu. 2017. SweepCanvas: Sketch-Based 3D Prototyping on an RGB-D Image. In *Proc. ACM Symposium on User Interface Software and Technology (UIST) (UIST '17).*
H Lipson and M Shpitalni. 1996. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design* 28, 8 (1996), 651–663.
Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 2017. 3D shape reconstruction from sketches via multi-view convolutional networks. In *IEEE International Conference on 3D Vision (3DV).* 67–77.
Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. 2019. PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
Fernando Naya, Joaquim Jorge, Julián Conesa, Manuel Contero, and José María Gomis. 2002. Direct modeling: from sketches to 3D models. In *Proceedings of the 1st Ibero-American Symposium in Computer Graphics SIACG.* 109–117.
Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. FiberMesh: designing freeform surfaces with 3D curves. *ACM Transactions on Graphics (Proc.*

SIGGRAPH) 26, Article 41 (2007). Issue 3.

Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. 2016. Interactive Sketching of Urban Procedural Models. *ACM Trans. Graph. (SIGGRAPH)* 35, 4, Article 130 (July 2016), 11 pages.

Patrick Paczkowski, Min H. Kim, Yann Morvan, Julie Dorsey, Holly Rushmeier, and Carol O'Sullivan. 2011. Insitu: Sketching Architectural Designs in Context. *ACM Transactions on Graphics* 30, 6 (2011).

Mengqi Peng, Jun Xing, and Li-Yi Wei. 2018. Autocomplete 3D sculpting. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.

A. Pipes. 2007. *Drawing for designers.* Laurence King.

Alec Rivers, Frédo Durand, and Takeo Igarashi. 2010. 3D Modeling with Silhouettes. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 29, 4, Article 109 (2010), 8 pages.

Robert McNeel & Associates. 2019. *Rhinoceros.* https://www.rhino3d.com/

Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009. Analytic drawing of 3D scaffolds. In *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, Vol. 28. ACM, 149.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. 2018. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. 2013. Geosemantic Snapping for Sketch-Based Modeling. *Computer Graphics Forum* 32, 2 (2013), 245–253.

Wanchao Su, Dong Du, Xin Yang, Shizhe Zhou, and Hongbo Fu. 2018. Interactive Sketch-Based Normal Map Generation with Deep Neural Networks. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018).

The CGAL Project. 2020. *CGAL User and Reference Manual* (5.0.2 ed.). CGAL Editorial Board. https://doc.cgal.org/5.0.2/Manual/packages.html

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. 2019. Learning to Infer and Execute 3D Shape Programs. In *International Conference on Learning Representations.*

Trimble. 2019. *SketchUp.* https://www.sketchup.com/

Yingze Wang, Yu Chen, Jianzhuang Liu, and Xiaoou Tang. 2009. 3D reconstruction of curved objects from single 2D line drawings. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D curve networks from 2D sketches via selective regularization. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 33, 4 (2014).

Pengfei Xu, Hongbo Fu, Youyi Zheng, Karan Singh, Hui Huang, and Chiew-Lan Tai. 2019. Model-Guided 3D Sketching. *IEEE Transactions on Visualization and Computer Graphics* 25, 10 (2019), 2927–2939.

Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. 1996. SKETCH: An Interface for Sketching 3D Scenes. In *Proceedings of SIGGRAPH (Computer Graphics Proceedings, Annual Conference Series).* 163–170.

Youyi Zheng, Han Liu, Julie Dorsey, and Niloy Mitra. 2016. SMART CANVAS : Context-inferred Interpretation of Sketches for Preparatory Design Studies. (2016).